Have No PHEAR: Networks Without Identifiers*

Richard Skowyra, Kevin Bauer, Veer Dedhia, Hamed Okhravi, and William Streilein
MIT Lincoln Laboratory
244 Wood St. Lexington, MA 02421
{richard.skowyra, kevin.bauer, veer.dedhia, hamed.okhravi, wws} @ll.mit.edu

ABSTRACT

Network protocols such as Ethernet and TCP/IP were not designed to ensure the security and privacy of users. To protect users' privacy, anonymity networks such as Tor have been proposed to hide both identities and communication contents for Internet traffic. However, such solutions cannot protect enterprise network traffic that does not transit the Internet. In this paper, we present the design, implementation, and evaluation of Packet Header Randomization (PHEAR), a privacy-enhancing system for enterprise networks that leverages emerging Software-Defined Networking hardware and protocols to eliminate identifiers found at the MAC, Network, and higher layers of the network stack. PHEAR also encrypts all packet data beyond the Network layer. We evaluate the security of PHEAR against a variety of known and novel attacks and conduct whole-network experiments that show the prototype deployment provides sufficient performance for common applications such as web browsing and file sharing.

1. INTRODUCTION

The essential network protocols that have enabled the success of the Internet were designed primarily with efficiency and performance in mind, not the security or privacy of users. Consequently, implicit and explicit identifiers can be found at all layers of the network stack and, in some cases, can be used to uniquely identify and track user activity. Even in cases where users trust their network operators, several recent high-profile anecdotes have shown that malicious third-parties can infiltrate network infrastructure (*e.g.,* routers or switches) for a variety of nefarious purposes including user monitoring, intellectual property theft, or network reconnaissance that can lead to secondary attacks within the network [8–10, 21, 24].

Removal or obfuscation of these identifiers while not overly disrupting normal network operation has been the primary focus of low-latency anonymity networks (most notably Tor [15]). The majority of these designs have a decentralized architecture that anonymizes communications by routing traffic through several geographically and topologically diverse intermediate hops, in order to to conceal the source and destination of the traffic. Such Internet-scale solutions, however, are not compatible with operating environments where internal network traffic may not transit the public Internet (such as an enterprise network). This paper proposes a system that enables anonymous communication within such networks via online elimination of all network identifiers that can be used to track users and restrict privacy.

Our Solution. We present the design, implementation, and evaluation of *Packet Header Randomization* (or PHEAR), a system that dynamically and transparently removes network identifiers such as persistent MAC addresses and IP addresses from packet headers, while still correctly routing network traffic. Our approach leverages emerging Software-Defined Networking (SDN) hardware and protocols that enable a fully programmatic interface to packet-forwarding within a network of SDN-enabled switches. PHEAR itself consists of end-host proxies that transparently replace packet identifiers with constant values or short-lived pseudonyms, a trusted SDN controller and management server that computes and installs pseudonym-based (as opposed to, *e.g.,*, IP-based) routes between hosts, and an infrastructure of untrusted SDN-capable switches—which could already exist at a target PHEAR deployment site—that forward identifier-free packets. Beyond hiding identifiers found in Layer 2 and 3 packet headers, PHEAR builds on existing standards such as IPsec for ensuring payload confidentiality and integrity and is compatible with recently proposed traffic analysis defenses [17, 35].

Evaluation. We critically evaluate the security and privacy provided by PHEAR with regard to a variety of known and novel attacks. We also conduct a whole-network performance evaluation of our prototype in an SDN network emulation environment with realistic user and traffic models. Our performance results indicate that PHEAR offers low-latency (time to first byte typically less than 500*ms*) that is sufficient for real-time interactive applications such as web browsing and high throughput (about 80% of the nominal throughput) to support bulk file-sharing applications.

Contributions. This paper contributes the following:

- First, we design PHEAR, an identifier-free system that dynamically eliminates identifying information from the MAC and Network layers through the use of emerging Software-Defined Networking protocols and hardware.
- Second, we implement a PHEAR prototype and critically evaluate its security against a wide variety of attacks.
- Third, we deploy our PHEAR prototype in a realistic whole-network emulation testbed. Our experiments show that the system's performance is sufficient to support typical applications including web browsing and file sharing.

Roadmap. The remainder of this paper is organized as follows. In §2, we motivate this work by summarizing the types of information leakage that can occur at various layers of the network stack; we also motivate our solution with an overview of existing SDN technologies. §3 presents the design, architecture, and implementation of PHEAR. §4 evaluates the performance and security of the system. Open questions related to PHEAR are discussed in §5 and this work is compared with prior work in §6. Finally, concluding remarks are given and future work is discussed in §7.

2. BACKGROUND

Since the core networking protocols that power the Internet were not developed with the security and privacy of users in mind, there are many opportunities for on-path network eavesdroppers to learn

*Distribution A: Public Release. This work is sponsored by the Department of Defense under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

information about communicating parties through the normal operation of networking protocols. However, with the emergence of new Software-Defined Networking (SDN) switches, there is a new tool suite available to system developers to improve the security and privacy of users without requiring a clean-slate re-design of the Internet's fundamental protocols.

In this section, we briefly describe the many forms of information leakage that can occur at all layers of the network stack. We also provide an overview of SDN, which is essential in our approach to eliminating the network identifiers.

2.1 Identifiers and Information Leaks

To enable the efficient transmission of data between devices on a network, a variety of network identifiers are commonly used to refer to unique hosts. While many of these identifiers are *explicit*, such as a physical device's MAC address or a long-lived static IP address, there are several types of *implicit* identifiers that can reveal information about a host on a network. We briefly describe several notable examples across multiple layers of the network stack.

Link-layer Identifiers. The data link layer is responsible for transmitting frames between hosts on the same 802 local area network (LAN). The MAC address is a six-byte value used to explicitly identify physical devices on an IEEE 802 network (such as Ethernet or Wireless Ethernet networks, for example). This address is a globally unique identifier for the device's network interface controller (NIC). The higher three bytes form the Organisationally Unique Identifier (OUI), which usually indicates the hardware manufacturer of the NIC; the lower three bytes form the NIC-specific portion. The source MAC address can be trivially observed by any switch on the same LAN as the sender.[1]

Network-layer Identifiers. The network layer, which is responsible for forwarding packets between networks, is a source of many explicit and implicit network identifiers. Examples include the following.

- **IP Address:** This identifier enables efficient routing of packets, and in some cases may be statically assigned to a host. In such cases, the IP address uniquely identifies the host.
- **IP Identifier:** The IP Identifier (IPID) is used for uniquely identifying the fragments of a datagram. While this field does not uniquely identify any host, the precise manner in which different operating systems generate IPID values has been shown to leak information about a host's platform, and widely used network mapping tools use IPID heuristics to fingerprint a remote host's OS [5].
- **IP Initial Time-to-Live:** All IP packets contain a TTL which is decremented at each router that the packet transits. The initial TTL value can vary by OS implementations. For example, Linux hosts use an IP initial TTL value of 64, while Windows hosts typically use 128 [4]. These implementation differences can leak information about a host's platform.

Transport-layer Identifiers. This layer is responsible for transferring data between running programs. TCP is known to contain several side channels that leak information about the identity of the host, including (but not limited to):

- **TCP Initial Window Size:** When a new TCP connection is established, the receiver's initial window size can reveal the host's platform, as this value has been shown to vary between Linux, Windows, and other platforms [4].
- **UDP and TCP Ports:** Services run on well-known destination ports by convention. For example, UDP packets destined for port 53 are usually DNS and TCP packets destined for ports 80 and 443 and HTTP and HTTPS, respectively. In fact, an Internet traffic study found that destination ports alone can be as effective as deep packet inspection for identifying the application-layer protocol of the traffic [36].
- **TCP Timestamps:** Unique patterns in clock skew that are remotely observable in TCP timestamps have been used to remotely fingerprint physical hosts [33] and find Tor hidden services [38].

Application-layer Identifiers. There are numerous examples of unique identifiers that can be found within application layer network traffic including e-mail addresses or instant messaging/VoIP login names. Such identifiers can be protected using security protocols such as SSL/TLS or HTTPS.

Despite the prevalence of identifiers across all layers of the network stack, there is currently no comprehensive solution that removes both explicit and implicit identifiers from network traffic. We propose a system that eliminates these identifiers by leveraging the capabilities of emerging SDN standards that enable customized protocol behaviors.

2.2 Software-Defined Networking

Software-Defined Networking (SDN) is an emerging suite of technologies to provide programmatic control over packet processing and routing. Since its inception in 2008 [37], SDN has coalesced into two distinct domains. *Network Function Virtualization* (NFV) [22] is based on decoupling network-layer services (*e.g.,* NAT, firewalls, caches, *etc.*) from the supporting hardware, permitting their virtualization and its accompanying benefits (*e.g.,* elasticity and migration). This approach relies on commodity switches supporting a software overlay network between virtualized network devices and end-host hypervisors. NFV is well suited to the cloud and datacenter domains, where the cost of replacing non-SDN network switches is high but virtualization of end-points and infrastructure is already well-supported. Furthermore, use of a software overlay necessitates packet encapsulation. The additional network overhead is less problematic in already high-capacity networks than in potentially bandwidth-limited enterprise environments.

In contrast to NFV, bare-metal SDN decouples the network control and data planes (rather than the network services and devices). The OpenFlow standard [40] is the primary protocol in this domain. It relies on a logically centralized, software-based controller, which communicates over a secure control plane to OpenFlow-enabled network switches. Because all SDN-related functions are performed purely in-network, OpenFlow does not require packet encapsulation or virtualization of network services or end-hosts. Instead, switches must support the OpenFlow protocol in firmware. Commodity switches in the network can still be used for traditional networking, but they cannot be managed by the controller.

We elected to use a bare-metal, OpenFlow-based SDN deployment in the design of PHEAR. Enterprise environments are expected to have both fewer and less-powerful switches than datacenters, with a commensurately lower cost of converting to SDN devices. Existing switches can be incrementally converted to OpenFlow switches as current infrastructure ages, or in some cases upgraded with only a firmware patch. Enterprises are also expected to have a higher cost associated with virtualizing network end-points, as these are spatially diverse (*e.g.,* individual desktops) rather than residing in a shared rack.

The OpenFlow standard [40] defines the required capabilities of compliant switches, as well as the communications protocol for interfacing with the controller. An illustrative example architecture is shown in Figure 1. An OpenFlow switch routes network data plane packets based on flow tables, which are ordered lists of

[1] Our work specifically addresses information leaks in wired Ethernet networks. See Pang et al. for a detailed analysis of information leakage in wireless LANs [41].

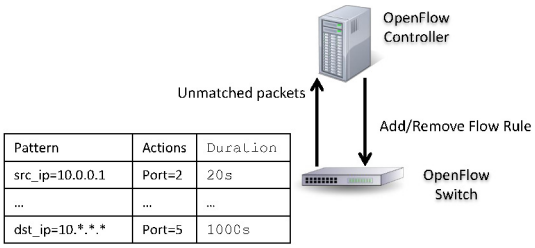


Figure 1: OpenFlow Architecture

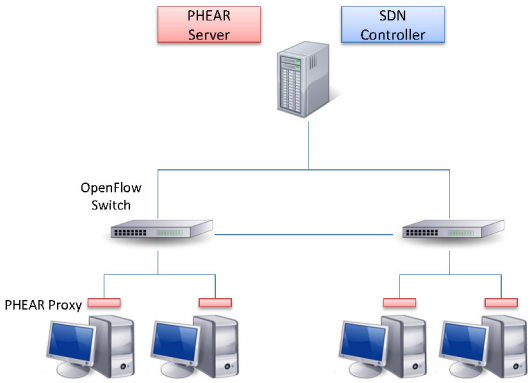


Figure 2: Packet Header Randomization Architecture

rules where each rule consists of a guard, a set of actions to trigger, and a time to expiration. The actions are activated and the packet processed only if the packet's header pattern-matches successfully against the guard for that rule. If a packet matches none of the rules in a flow table, it is forwarded to the OpenFlow controller. If a packet matches multiple rules, the first rule encountered sequentially is triggered and any remaining are ignored. Supported actions include forwarding to an egress port, rewriting header fields, matching against another flow table, and assigning aggregation tags to packets in order to form traffic groups. A pattern-match guard can use either explicit field values (*e.g.,* `srcIP=192.168.1.1`) or mix in wildcards (*e.g.,* `srcIP=192.168.1.*`).

Since OpenFlow is implemented in firmware and must operate at line rate, some functionality is unavailable. There is no support for higher-order matching of packet headers, such as regular expressions. Actions are limited to a set of operations (see [40] for a full list) over packet headers only. No support for whole-packet processing, cryptography, or other computationally intensive functionality is present. This prevents, *e.g.,* SDN-based onion routing from being implementable on OpenFlow switches.

Communication between the OpenFlow switch and controller takes place over a secure tunnel from the switch's management port to the machine running the controller. Switches forward to the controller packets which either do not match any flow table rule or which have an explicit action to that effect. The controller generates and/or removes flow rules in one or more switches in response. The actual implementation and functionality of the controller is determined completely by the application domain, but all controller programs must communicate with switches only by installing flow rules. Note that does not restrict the controller from communicating with other machines via a northbound API. Controllers can be written in a general-purpose language (*e.g.,* Python, Java) or one of a number of languages designed for the purpose. These include NOX/POX [23], Beacon [18], Frenetic [19], and Flowlog [39].

3. PHEAR

Here we present an overview of PHEAR, a system designed to provide unlinkability of traffic source and destination at Layers 2 (MAC) and 3 (Network) of the network stack. We first identify our assumptions and threat model, and then describe the design and implementation of the PHEAR protocol.

3.1 Assumptions and Threat Model

The design of PHEAR is motivated by the growing incidence of long-term, stealthy, and pervasive attacks on enterprise networks by dedicated attackers, referred to as an Advanced Persistent Threat (APT) [8]. Recent examples of such attacks include instances of an APT attacker not only compromising end-hosts, but spreading into the network infrastructure itself in order to facilitate network reconnaissance and persistence [9]. Compromises have also been observed against home and small business routers [10,21,24]. Finally, evidence has been found in recent years of supply-chain attacks on commercial routers, in which backdoors have been inserted during manufacturing or distribution. These enable attacks against network infrastructure even when vulnerable administrative services are not exposed. Once in the network, such attackers can eavesdrop on private communications, steal private data and intellectual property, and launch additional attacks against other hosts.

We consider a scenario in which an enterprise network (LAN or WAN) is targeted by an APT-type adversary who has compromised one or more pieces of the network infrastructure (*e.g.,* routers or switches). From this position, the adversary can monitor and manipulate traffic flowing though their compromised switches. Our main objective is to hide both the identities of communicating parties and the contents of the communications. This provides *unlinkability* (in the terminology of Pfitzmann and Hansen [42]) of communications from the perspective of these switches. Since switches can always observe the ingress and egress ports used by a packet, this unlinkability is limited to the *anonymity set* defined by that port: the set of all possible origins and destinations given the subnets defined by those switch ports. In the case of compromised edge switches the anonymity set cardinality may be one (that is, only a single end-host is reachable from a particular ingress or egress port). In this case PHEAR provides little benefit. The impact of network topology on security is considered in depth in §4.2.

PHEAR's threat model is similar to that of Tor [15]: the adversary may monitor some fraction of network traffic, and can modify, replay, reroute, drop, delay, or inject packets in-flight. Additionally, the adversary can also observe the routing tables in compromised switches. We do not claim protection against a global adversary, who may view all network traffic, or to end-to-end traffic confirmation attacks [34,46]. Similar to existing low-latency anonymity networks, PHEAR is vulnerable to traffic analysis attacks that leverage side channels including packet size and timing (*e.g.,* website fingerprinting [48] and protocol identification [52]); although in §4.2 we discuss certain parameters and configurations of the system that may decrease the accuracy of such traffic analysis attacks.

The PHEAR threat model differs from other anonymity networks in one significant respect: since the users of the network belong to the same organization that operates it, the network *provider* is trusted in our model but individual *infrastructure* components are not. Since the system relies on SDN, we trust the network controller but we do not trust the individual switches.

3.2 Design

Overview. Unlinkability of a message to its source or destination is a crucial privacy requirement when considering adversaries who can observe or manipulate traffic. Tor provides this property on an Internet scale via an overlay onion routing network. On the enterprise scale, however, the situation is somewhat different. Large numbers of end-hosts cannot be leveraged as effectively, and routing hardware lacks the ability to perform any of the cryptographic operations necessary for onion-based schemes. PHEAR is an attempt to address this gap and provide unlinkability in enterprise networks through a combination of packet header rewriting and Software-Defined Networking.

Unlinkability is provided by normalizing all layer 2 and 3 header fields except source and destination IP addresses. Higher-layer headers can already be encrypted through IPsec [32], which we deploy alongside PHEAR to provide full coverage of the packet header. Rather than using IP addresses (which trivially leak identity) to route traffic, PHEAR utilizes OpenFlow-based SDN to route using temporary 64-bit nonces, which are stored in the IP address fields of a packet header and are generated independently of the packet source or destination. These nonces last for a configurable duration, and allow an ongoing connection to be routed between endpoints without leaking (to the network switches) any endpoint identifiers.

The architecture, which consists of three primary components, is shown in Figure 2. On a high level, end-host proxies mediate access to the network, rewrite packet headers (to normalize identifying fields and insert temporary routing nonces), and request and expire pseudorandom identifiers. The PHEAR server manages the current pool of routing nonces, generating and removing them as required. Finally, the SDN controller maintains a mapping between routing nonces and end-host locations, which it uses to generate on-demand flow rules for the (untrusted) switches. These roles are elaborated below.

Performance Goals. Minimizing impact on a user's perceived latency and throughput is a critical factor in encouraging adoption of PHEAR. Our base system imposes a latency cost whenever an endhost requests a new routing nonce for a session. This occurs at least once, at initiation of a new connection, and may occur more frequently depending on each end-host's configuration. While a nonce is in use, no additional delay is added to an ongoing session. Throughput is impacted primarily by PHEAR endpoint proxies, which perform per-packet header rewriting. We experimentally assess the performance impact of our system on typical traffic (*e.g.,* interactive, web, and bulk downloads) in §4.1.

Security Goals. Our primary security goal is unlinkability of a message to either its origin or destination beyond the set of possible origins (destinations) indicated by the packet's ingress (egress) port. Note that in cases where a malicious edge switch observes a packet, one or both of these set sizes may be a single end-host. Thus, certain network topologies (*e.g.,* a single-switch star) may have little to gain from deploying PHEAR. In addition, we do not provide unlinkability at the flow level: two packets belonging to the same ongoing connection will (usually) have the same header and be linkable to one another. System configurations in which nonces are frequently refreshed weakens flow-level linkability, and may inhibit traffic analysis at the cost of increased latency. We investigate the security of both configurations in §4.2.

Non-Goals. In addition to the security and performance goals above, there are several non-goals which we explicitly do not address, either because their perceived benefit does not justify the cost, or because they are open problems outside the scope of this work.

Concealing PHEAR: Due to the sweeping changes made by PHEAR to network packet headers, as well as the wide variety of IP addresses in use (including those which would not normally be routable, or reserved for special usage), it will likely be apparent to an attacker that PHEAR is in use. An attempt could be made to mitigate this by, *e.g.,* limiting the possible space of nonce values, refreshing nonces only infrequently, and randomizing (rather than normalizing) header fields. The performance and security tradeoffs are unlikely to be worthwhile, however, given that it is not clear what the attacker gains by being aware of PHEAR or that these measures would be enough to stop PHEAR deployment from being detected in the first place.

Unobservability: While unlinkability prevents an attacker from linking a message to a specific sender or receiver, observability prevents the attacker from knowing a message has been communicated at all [42]. PHEAR does not (and is not intended to) provide this property. The authors are unaware of any system which provides this property without substantial cover traffic overhead, which would not be suitable for an enterprise network.

Statistical Traffic Analysis: Attackers which use statistical or machine learning techniques to correlate non-header packet features (*e.g.,* size, total session traffic volume, inter-arrival time, *etc.*) in order to infer protocol in use, fingerprint websites, *etc.* are outside the scope of our threat model and an open problem in low-latency anonymity networks. Arguably, the intent of PHEAR is to force an adversary to mount this kind of attack by removing the trivial-to-analyze packet header identifiers.

3.3 Architecture

In this section, we discuss the architecture of PHEAR. Specifically, we describe each system component in detail, the cryptographic requirements we impose on end-to-end encryption schemes deployed alongside PHEAR, and the protocols used to establish and maintain unlinkability.

3.3.1 System Components

PHEAR has three primary components: a proxy which performs packet header rewriting and resides on end-hosts, a server which manages anonymous routing nonces, and a controller which implements packet-handling rules on SDN switches.

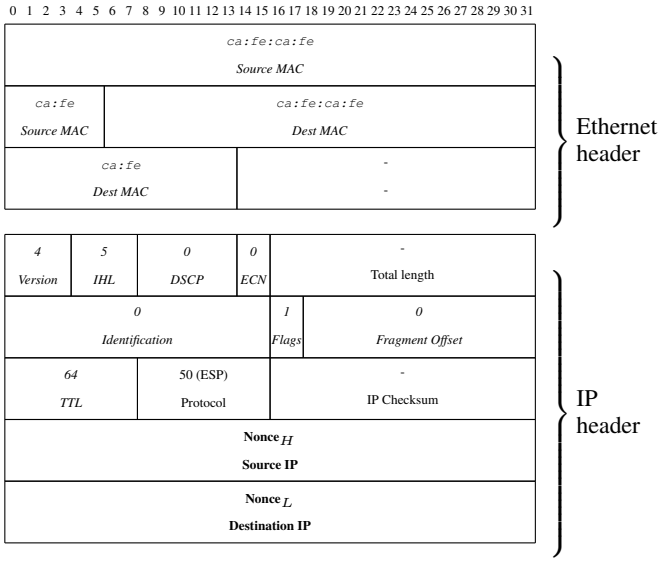


Figure 3: PHEAR Header

PHEAR Proxy. The PHEAR proxy runs on all participating end-hosts, and is responsible for transparently rewriting incoming and outgoing packets in order to remove (or replace) explicit or implicit identifiers. Figure 3 presents a PHEAR packet header, which is representative of all protected packets on the network. The two bolded IP address fields in the figure are used by PHEAR to store a 64-bit nonce, which is treated as a temporary routing token by the SDN switches. Italics in the figure denote fields which are set to constant, normalized values in order to remove any identifiers that can be used in fingerprinting of operating systems, protocols, *etc.*. Error-detection fields (total length, checksums) are updated to correct values but not otherwise modified.

When a packet is received from the operating system (*i.e.,* is outgoing), the proxy first checks if its header has a locally cached nonce associated with it. If not, it queries the server over a secure channel (see 3.3.3), sending the original header and receiving a PHEAR nonce in response. Once a cached copy is present, the proxy replaces the original headers of all packets on the flow with the obfuscated variant before sending it to the network. When a packet is received from the network, the proxy replaces the PHEAR header with the actual packet header, which is either cached locally or (if the flow has not been seen before) retrieved from the server.

The PHEAR proxy is also tasked with refreshing local nonces. The rate at which nonces are refreshed is referred to as the *nonce refresh rate*, and is typically on the order of several seconds (*e.g.,* 30 or 60 seconds). Existing network connections can get a fresh nonce assigned by requesting a new value from the server. This expires the old nonce and re-maps the ongoing network route to the new value. Each proxy can choose its own nonce-refresh rate, which represents a tradeoff between latency and security. Long-lived nonces impose minimal performance impact, but short-lived nonces may hinder traffic analysis by weakening likability of packets to ongoing connections. This tradeoff, as well as the possible consequences of uniquely identifiable nonce refresh rates, is explored in §4.2.

Finally, the proxy maintains a last-seen time for each locally cached nonce. If no traffic associated with that nonce arrives before a tunable threshold is crossed, the flow is considered inactive and can be terminated. The proxy notifies the server, and both drop the mapping from their databases.

PHEAR Server. The server component of PHEAR is a control-plane process that manages the global mapping of actual packet headers to PHEAR nonces. The server ensures that this mapping is one-to-one with respect to the current pool of active connections: each unique packet header is associated with exactly one 64-bit routing nonce. Nonces are generated using a collision-resistant hash function (*e.g.,* SHA-1) with an input of the actual source and destination IP addresses concatenated with a salt value whose size accounts for the remaining bits of the function's output range (*e.g.,* a 96-bit salt for a 160-bit SHA-1 value). The nonce may change if it is refreshed by a proxy, but will never map to greater or fewer than one packet header. When proxies expire a nonce without requesting a new value, the server treats it as connection termination and removes the mapping from the database.

In any case in which a nonce mapping changes or is removed, the server first confirms that the nonce is associated with the requesting proxy. IPsec channels are bound to an identifying certificate. If that identity is also associated with the originating endpoint of the routing nonce, the transaction is allowed. Additionally, whenever a nonce is removed from use, the server informs the OpenFlow controller of the expiration so that flow rules associated with that value can be expired.

Due to the sensitive nature of communication between the server and proxies, all messages are sent over a cryptographically secure channel to prevent network eavesdroppers from trivially deobfuscating packet headers. While existing protocols (*e.g.,* IPsec) can (and are) used for this purpose, establishing an authenticated channel without leaking identifying information (by *e.g.,* sending certificates over the wire) to an eavesdropper requires care. We discuss the requirements on authentication below, and the security implications in §4.2.

OpenFlow Controller. PHEAR relies on Software-Defined Networking (SDN) to route over nonces instead of IP addresses, and is implemented using the OpenFlow [37] SDN standard due to OpenFlow's widespread support by network equipment vendors. The controller is responsible for managing global routing state by adding and removing flow rules from each switch's flow table. The network topology is stored here, with individual end-host identities (represented by certificates rather than IP addresses) bound to edge switch ports.

When the controller receives an unhandled PHEAR packet from a switch, it queries the server database with that packet's nonce in order to identify the intended endpoints. It then computes the next hop for that packet and emits a flow rule for handling that nonce in the future. The rule defines a pattern match over the 64-bit nonce identifier, with a trigger action forwarding packets to the correct egress port. This approach ensures routing correctness, while simultaneously concealing the endpoint of a communication from intermediate switches by routing only over temporary nonces.

The controller also receives nonce expiration messages from the server. Since the controller does not maintain a local copy of each flow table, it is not aware *a priori* which switches have flow rules associated with the expired nonce. Rather than querying each switch and imposing overhead on expiration, the controller simply broadcasts flow rule removal updates to all switches. If the rule is not present, the switch ignores the update. Note that this informs an adversary whenever a nonce is no longer in use. We consider the security implications of this in §4.2.

3.3.2 Cryptographic Requirements

In order to ensure nonce uniqueness and maintain correct routing state, unprotected packet headers must be exchanged between PHEAR proxies and the server. These messages are routed over untrusted switches, and so must be cryptographically secured in order to ensure that an eavesdropper cannot map nonce values to unprotected packet headers.

Restricting identifiers from being transmitted over the network imposes requirements on any cryptographic protocols used to create secure channels. Namely, no scheme can be used which implicitly or explicitly transmits plaintext linkable to a particular host or set of hosts. Examples of such leaks include digital certificates and PKI-based digital signatures. In the former case identity is trivially leaked to an eavesdropper. In the latter, an attacker can simply iterate over the public keys used in the network. The key which verifies the message is authentic could then be used to identify the message originator. This disqualifies, for example, TLS [14] as a drop-in solution for providing secure channels.

Fortunately, IPsec in Transport Mode [45], with Security Associations set up using Internet Key Exchange (IKEv2) [30], does suffice as a drop-in solution for secure channel setup. IKEv2 is a two-phase protocol, and uses an initial Diffie-Hellman exchange to set up a confidential channel over which certificates can be exchanged securely. Once identity is verified with public-key operations, shared symmetric keys are derived for future encryption and authentication. These keys are not linkable to a specific endpoint,

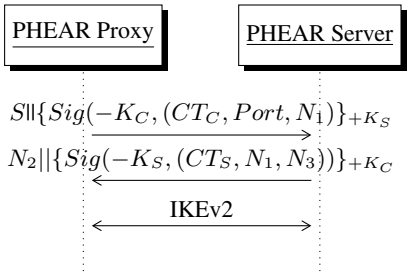


Figure 4: Client Registration

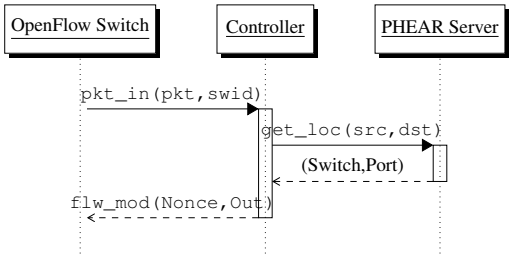


Figure 5: Nonce Lookup

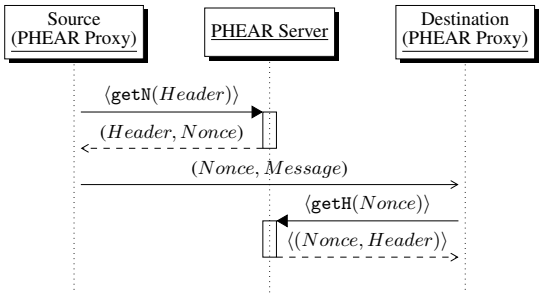


Figure 6: PHEAR Session Initiation

as they are generated pseudorandomly and contain no identifying information.

In addition to proxy-server security, IPsec is also highly complementary to PHEAR's primary goal. Recall that PHEAR only protects layers 2 and 3 of the networking stack. Transport-layer and application data are rich sources of identifying information (*e.g.,* TCP ports), and must also be secured in order to ensure unlinkability. IPsec provides this additional coverage: when both PHEAR and IPsec are deployed all explicit and implicit identifiers originally present in packet header fields are removed.

3.3.3 PHEAR Protocols

In this section, we discuss the protocols used by PHEAR to provide unlinkability and maintain routing correctness.

Proxy Registration. When a new host is added to a PHEAR network, its proxy must establish a secure channel with the server in the presence of possible adversaries. IKEv2 cannot be deployed immediately, as the host's network location is not yet known to the controller and it has no distinct identifier with which to originate traffic. The registration protocol in Figure 4 addresses this issue, making use of the certificates already present due to the deployment of IPsec. The PHEAR server is always reachable from at a well-known 64-bit identifier S, analogous to a static IP address. Since this prevents unlinkability of the message destination, however, S is used only once during the registration protocol.

The newly joined proxy sends a message containing its certificate (CT_C), the port it is attached on, and a nonce N_1 used to verify that the message was read by the server. The message is signed by the proxy's private key and encrypted with the server's public key. Note the order of operations: if we instead chose to encrypt then sign, an eavesdropper could determine the message sender by iterating over the public verification keys.

On receipt of a registration message, the server adds a mapping in its database from a fresh nonce N_2 to communications originating at the server and terminating at the network port bound to the proxy's certificate. It also adds a nonce N_3 to the database for messages going from the proxy port to the server, which removes the need to route on S and leak information. The server responds with verification nonce N_1 as well as routing nonce N_3. Note that N_3 also provides replay protection: an attacker replaying the proxy's initial message will not know the nonce to use for future communications. N_2 and N_3 can then be used for each direction of the IKEv2 handshake. Once keys have been derived, IPsec in transport mode is deployed for the duration of network operation.

Nonce Lookup. Nonces in PHEAR are used to route traffic between two endpoints without revealing the identity of those endpoints to intermediary switches. This capability leverages OpenFlow's ability to dynamically update switch flow tables in response to the receipt of a packet for which no existing rule exists. Figure 5 illustrates the protocol used whenever this occurs. The switch receiving an unknown packet forwards it, as well as the switch's identifier, to the controller. There, the 64-bit nonce is extracted from the source and destination IP address fields. This value is used to query the PHEAR server, which responds with the network location (*i.e.,* switch and port) associated with the destination endpoint for that nonce. The controller computes the necessary egress port given the destination, and emits a flow rule to the switch mapping the nonce to that port. The flow rule persists until it is removed by the controller due to nonce expiration.

Note that the above protocol operates on a per-switch basis, rather than per-route. This allows the controller to maintain a static, ternary table mapping destination identity and switch identifier to an egress port and have a minimal computational burden associated with each lookup. In turn, the latency imposed by forwarding the packet to the controller and waiting for a response is proportional to the number of switches along a path. The time/space tradeoff could be reversed by storing the topology as a graph, computing the shortest path on a per-nonce basis, and installing the entire route at once. This would make each lookup take at least linear time; however, it potentially opens the controller to denial-of-services attacks from forged nonces.

PHEAR Session Initiation. Figure 6 shows the protocol for setting up a new PHEAR session. Angle brackets in the figure denote messages sent over the secure (*i.e.,* confidential and authenticated) channel set up during proxy registration.

First, the originating end-host sends the actual packet header to the PHEAR server over a secure channel and requests a new routing nonce. The server generates a 64-bit pseudorandom value and stores the mapping between the nonce and header, then returns the nonce value to the message sender. The sender caches this value for the duration of the session. Once the session terminates (*i.e.,* the activity timer expires), the proxy will notify the server and can delete the nonce from its cache.

Once the routing nonce is cached at the originator's PHEAR proxy, it can begin sending identifier-free packets (whose headers contain only constant values and the 64-bit nonce) to the receiver. This process informs any switch along the packet's path what ori-

gin and destination subnets (as defined by ingress and egress port) the actual origin and destination hosts are part of, but provides no other explicit or implicit identifiers in the packet header which can be used to constrain these anonymity sets.

When the receiver gets the first packet in a new PHEAR session, it requests the actual header from the PHEAR server over a secure channel and caches it locally. Before responding, the server confirms that the receiver's identity matches the destination IP address in the actual packet header. As the channel being used for this exchange is authenticated, this reduces to confirming that the channel is associated with the certificate owning that IP address.

The adversary's view of this protocol is limited to messages seen by the network switches. In the case of exchanges with the PHEAR server (which contain original packet headers and mappings to nonces), these messages are encrypted and authenticated. In the case of exchanges between endpoints, message payloads and the transport layer are encrypted with IPsec. The network and datalink layers are stripped of identifiers by PHEAR. This forces an eavesdropper to rely on traffic analysis of side channels (*e.g.,* packet timing, size, and volume) rather than direct observation. PHEAR does not provide any security guarantees with respect to traffic analysis, but we believe that a high nonce refresh rate may impede correlation of packets to specific ongoing sessions (especially in high-bandwidth settings).

3.4 Implementation

Our prototype is implemented and tested in an emulated OpenFlow SDN environment. This enabled rapid, automated scaling and reconfiguration of network topology. The SDN controller is implemented in Flowlog [39], a tierless declarative language for building verifiable OpenFlow controllers. We chose Flowlog in order to minimize the attack surface of the SDN controller: if switches can be malicious, they may attempt to subvert the controller over their dedicated link. Flowlog controllers compile to both an executable and a specification in Alloy [26] which enables formal verification of controller behaviors. Flowlog also supports Northbound APIs via event-based and remote table abstractions, which utilize Apache Thrift RPC to define remote interfaces. We utilize these to enable communications with the PHEAR server.

The server is written in Python and uses Apache Thrift to define remote interfaces with both the OpenFlow controller and PHEAR proxyies. SQLAlchemy is used to interact with a back-end SQLite nonce database. In our prototype, the server is co-located with the controller, but it could easily be moved to any routable system.

Finally, the PHEAR proxy is written in C in order to use the Netfilter Queue library with `iptables` to incercept packets queued in the Linux kernel and manipulate them in user-space. This enabled rapid development and prototyping of the proxy without having to write Linux kernel modules.[2]

4. EVALUATION

In this section, we quantify the performance and security of PHEAR through experimentation with our prototype implementation in a realistic network emulation testbed.

4.1 Performance Evaluation

Experiment Setup. We deploy our system on an enterprise-like network in Mininet shown in Figure 7.[3] We conduct whole-network

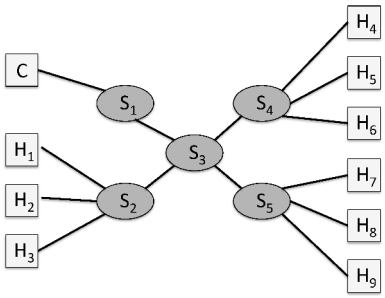


Figure 7: This emulated network topology consists of five switches (S_1–S_5), nine end-hosts (H_1–H_9) and one SDN controller (C). Each full-duplex link is configured with 1 ms latency and 100 Mbit/s bandwidth.

experiments with two distinct classes of users: interactive web browsers and bulk downloaders. The web browsing users fetch 300 KB files over HTTP, pausing for 1–10 seconds (uniformly distributed) between fetches. The bulk downloaders fetch 5 MB files, pausing for 1-5 seconds (uniformly distributed).[4] The PHEAR server and controller are run on host C and each end-host (denoted H_n) runs a local instance of the PHEAR client. Hosts H_1–H_3 serve files over HTTP, H_4 and H_7 are bulk downloaders, and the remaining hosts are web browsing users.

Our experiments compare PHEAR's performance to that of point-to-point IPsec (tranport mode) as a baseline. Additionally, we also measure the performance impact of the nonce refresh rate parameter. We conduct experiments in which nonces are refreshed every 30 s, 60 s, 120 s, and no nonce refresh.

Metrics. We measure performance from the user's perspective with two standard metrics from the anonymity network performance literature [28, 47]. First, we measure *time-to-first-byte*, which is the amount of time between a client's request for a TCP connection and receipt of the first byte of data from the server. This measure is particularly relevant to real-time applications such as web browsers and live streaming video, where responsiveness is important. Second, we measure *total download time*, which is simply the total time necessary to receive the last byte of data. This metric is important for applications where high throughput is valuable (e.g., bulk downloads or file sharing).

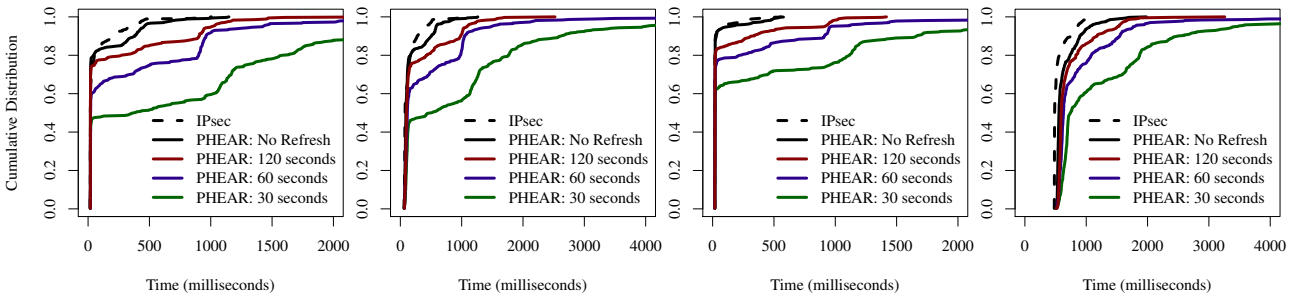
Results. Our whole-network performance measurements are summarized in Figure 8 as cumulative distribution functions (CDFs) of the time-to-first-byte and total download time for the emulated web and bulk clients.

The time-to-first-byte for web and bulk clients, shown in Figures 8(a) and 8(c), is similar for PHEAR (with no refreshes) and IPsec. Both protocols require the assistance of the SDN controller to install flow table forwarding rules at each of the switches along the path from the sender to the receiver, but this operation only occurs once for each communicating client/server pair. PHEAR clients do, however, register each client/server pair with the PHEAR server to obtain a nonce, which requires an additional round trip at both the sender and receiver to obtain and resolve the nonce. Since the experimental network has only three distinct servers, in this experiment only three nonces are requested for each client (one for each client/server pair). Now as the nonce refresh rate increases

[2] Alternatively, the proxy could be implemented fully in kernel space to avoid the performance impact of kernel/user context switching. However, this comes with additional deployment obstacles, as all users would need to run a custom kernel.

[3] Due to the sensitive nature of an organization's internal network layout, to the best of our knowledge no data sets are available on real-world enterprise network topologies and configurations. As such, we make reasonable modeling assumptions about network shape, host placement, number and capacity of switches and links, and other important configuration details.

[4] This traffic generation approach is similar to prior work in emulating users of anonymity networks [28, 47].



(a) Web time-to-first-byte (b) Web download time (c) Bulk time-to-first-byte (d) Bulk download time

Figure 8: Whole-network performance results for the emulated web and bulk clients.

from once every 120 seconds to every 30 seconds, we observe the tail of the distributions increase. These additional delays are the result of additional nonce requests at the source and destination hosts, and the additional forwarding table rules to be installed at each switch along the traffic's path. Given the relationship between refresh rate and network delay, we recommend that the refresh rate be higher than 30 seconds (for this experimental network). But despite these delays, we believe that the absolute performance cost is modest in terms of impact on the user's quality-of-service.

Figures 8(b) and 8(d) show the distribution of total download times for the web and bulk clients, respectively. Relative to IPsec, web clients that use PHEAR (with no refreshes) experience very similar performance. However, the bulk clients' download times do incur a modest penalty. This penalty is due to the user-space packet interception and dynamic header rewriting by the PHEAR clients, which operate at the sender and receiver. In addition, we note that the relatively high time-to-first-byte is a significant component of the total download times as the nonce refresh rate decreases. As such, we note that the nonce refresh rate is an important parameter for total download time as well as time-to-first-byte. But the performance cost is still reasonable for most users: for example, 90% of the PHEAR web requests with a fast 30-second nonce refresh rate complete in less than three seconds; 85% of the bulk requests with the same refresh rate complete in less than four seconds.

Overall, these results demonstrate that, under a realistic workload of web and bulk file sharing traffic, the PHEAR prototype offers sufficient performance to support expected user traffic. Furthermore, PHEAR offers similar performance to IPsec alone when the nonce refresh rate is sufficiently high.

4.2 Security Evaluation

In this section we summarize several classes of attack that an adversary might mount on PHEAR, and how our system is impacted.

4.2.1 Traffic Analysis Attacks

PHEAR does not provide any strong guarantees about adversarial traffic analysis. Indeed, by stripping identifiers from packet headers, the goal of PHEAR is arguably to force an adversary into using such attacks rather than being able to trivially link endpoints to messages.

Session Linkability. A necessary pre-condition for (to the best of our knowledge) all current traffic analysis techniques is correlating observed packets to ongoing sessions. When identifiers like IP addresses are available this is trivial. However, PHEAR routing nonces complicate the task, especially when refreshed at a high rate and aggregated with other ongoing sessions from both the same end-host and others sharing that switch port. To link multiple nonces to the same session requires an initial statistical analysis step, prior to actually drawing conclusions about the traffic itself.

Statistical Fingerprinting. Analysis of identifying characteristics not present in the header, such as inter-arrival times, packet size, and volume is outside the scope of our threat model. In quiescent network, an attacker able to observe traffic to and from the PHEAR server (*e.g.,* by compromising its edge switch) may be able to link nonce requests with new traffic. The nonce values are not themselves visible in the exchange, but are correlated in time to new nonce installations on switches.

End-to-End Correlation and Watermarking. Unlike onion routing, PHEAR packet identifiers do not change each hop. Thus, there is little need to watermark or correlate end-to-end traffic: packets are linkable to an ongoing connection simply by sharing the same routing nonce. These attacks may be more useful in settings where the nonce refresh rate is high and session linkability is nontrivial.

Transport Layer Analysis. PHEAR does not strip identifiers from the transport layer, as it is designed to be deployed alongside a protocol granting transport-layer confidentiality and authentication, such as IPSec in transport mode. These protocols are not designed for anonymity of end-points, however, and may leak identifying information. Internet Key Exchange (IKE) handshake implementations and parameters can be used to fingerprint operating systems, for example [25]. IKE is frequently used with IPSec to establish shared keying material, and thus may leak identifiers via plaintext handshake payloads even when packet header fields are secured with PHEAR. Furthermore, IPSec uses monotonically increasing sequence numbers, which might be used by an attacker to correlate a newly refreshed nonce with its prior value. PHEAR could be extended into the transport layer to mitigate such header-specific leaks, but in doing so would bind itself to a specific transport layer security protocol.

4.2.2 Linkability Attacks

Attacks in this section are fundamentally concerned with reducing or eliminating unlinkability of message origin and destination without resorting to statistical traffic analysis.

Targeted Switch Compromise. An adversary interested in linking specific hosts can do so by compromising the edge switches used by those hosts. While PHEAR still guarantees that linkability is limited to ingress/egress port anonymity sets, at least one of these

sets has only a single member on edge switches. The size of the remaining set is topology-dependent (or one if both hosts are on the same switch). Hierarchical networks enforce a tradeoff between anonymity set size and coverage of network traffic, for example. Conversely, single-switch star topologies gain no benefit from deploying PHEAR.

Multiple Switch Compromise. If more than one switch in a network is malicious, then the anonymity sets provided by PHEAR will be the smallest sets seen by any switch on a route. That is, the origin set will be the set observed at the ingress port of the switch closest to the sender, and the destination set will be the set observed at the egress port of the switch closest to the receiver.

Server Compromise. The nonce server is a trusted system component which maintains identity mappings for all current nonce values. If it is compromised, communicating end-hosts become linkable to one another. Unless expired mappings are stored by the network operator for diagnostic reasons, an adversary compromising this server would not be able to link the endpoints of previous communications.

Controller Compromise. If the SDN controller is compromised, it can both pull the existing identity mappings from the server, as well as form a global view of the network by querying switches for their flow rules. In both cases linkability is broken. One difference from the server, however, is that the controller can be made verifiably secure against certain classes of vulnerability by using SDN programming languages such as Flowlog [39] which output both executable code and formal models of that code. While this does not guarantee the absence of all vulnerabilities, it reduces the available attack surface by eliminating classes of exploitable bugs.

End-Host Compromise. An attacker operating a compromised end host trivially learns the identities of machines connecting to it. While PHEAR could be configured to not reveal the origin of connections (*e.g.,* by not restoring the source IP field of the packet header), doing so will likely break any ongoing application-layer sessions (*e.g.,* streaming media) if the sender refreshes nonce values and the sender IP changes. We elected to minimize our impact on user experience, rather than provide origin anonymity to the destination.

Control Plane Man-in-the-Middle (MiTM). Traffic from clients to the server can be linked to the server during client registration, due to the static identifier used by the server for bootstrapping new clients. After this point, communication is routed over standard PHEAR nonces. Also during client registration, an adversary may attempt to MiTM the identity verification handshake prior to IKEv2 being deployed. As messages in both directions are signed and encrypted, the adversary is limited to replay attacks rather than corrupting or forging messages. If the message originating from a client is replayed, however, the attacker will be unable to read the routing nonce to use for future communications. If the message originating at the server is replayed, the verification nonce will not match and the client will abort. Once the IKEv2 handshake begins, we rely on the security of IPsec to prevent attackers from decrypting, forging or corrupting client-server communications.

Data Plane Man-in-the-Middle. Switches can drop, corrupt, forge, or redirect traffic. In terms of linkability, normal data plane traffic is confidential and authenticated (via IPsec) as well as stripped of identifiers. Authentication must be implemented via message authentication codes (*e.g.,* using IPsec), however. Digital signatures can leak identifying information by allowing the attacker to iterate over the public verification keys of all network participants.

Randomness Exhaustion. If the nonce server has access to a source of randomness which is replenished slower than the total rate at which new nonces are requested, then future nonce values may become predictable. Rate-limiting of nonce requests can mitigate this issue, but may have a performance impact in large networks.

Nonce Expiration Notification. The SDN controller does not maintain a local copy of the global routing state, and thus must broadcast flow rule expirations for specific nonce values to all switches in the network. This notifies adversarial switches as well. The broadcast does not directly impact linkability, as the set of current nonce values is not itself sensitive. However, it may make it easier for an adversary to distinguish is a particular nonce is associated with a single long-duration connection of multiple short connections broken by nonce expiration broadcasts. Nonces are drawn pseudorandomly from a 64-bit space, however. For a specific nonce value to recur frequently, a substantial amount of network traffic would be required.

Nonce Refresh Rate Analysis. PHEAR nonces are refreshed at a client-determined rate in order to allow each end-host to tailor the protocol for its own workloads. However, the rate at which nonces are refreshed is itself a temporal identifier, and could be used to be used to fingerprint end-hosts. To mitigate this end-hosts can be configured to use a universal refresh rate, or can introduce noise into their individual refresh rates in order to obscure the timing signal. The former choice eliminates the identifier entirely, but may negatively impact performance (if the refresh rate is high) or security (if it is low).

4.2.3 Availability Attacks

This section considers attacks on resource availability, in order to disrupt or deny legitimate users. Note that stopping such attacks is not a primary goal of PHEAR, and successfully mounting one does not compromise unlinkability.

Nonce Exhaustion. If an end-host continually requests new nonces without ever expiring them, the total pool of available nonces will necessarily be depleted. This behavior is easily distinguishable from normal usage patterns, however. In such cases the server could expire the nonces on its own.

Cryptographic Denial of Service. The client registration protocol uses public key cryptography in its identity verification handshake. An attacker could exploit this by continually replaying captured client messages or forging its own, forcing the server to decrypt and verify the message before discarding it.

Spurious Nonce Denial of Service. If spoofed messages containing spurious nonce values are rapidly sent to the network, the controller could be overwhelmed with (bogus) nonce lookup requests. In this case, new, legitimate lookup requests would be dropped. Existing flow rules would remain, however, so ongoing sessions are unlikely to be disrupted.

Packet Dropping. Malicious switches can trivially choose to drop or redirect packets passing through them. PHEAR does not mitigate this, but also does not amplify the effect.

5. DISCUSSION

In this section, we discuss many open questions related to the design and operation of PHEAR.

5.1 Traffic Analysis Resistance

Typically leveraging features that are preserved in encrypted network traffic such as packet size or inter-arrival time, modern traffic analysis attacks often employ statistical or machine learning techniques to infer sensitive information about the contents of encrypted network traffic. Examples of such attacks include website fingerprinting [29, 48, 49], web browser fingerprinting [53], identi-

fication of spoken phrases in encrypted VoIP conversations [50,51], and inference of the application protocol [52].

While the primary goal of PHEAR is not to defeat these types of sophisticated attacks, we believe that the use of multiple nonces for each underlying flow may reduce the accuracy of these statistical attacks. Furthermore, our design is compatible with existing traffic analysis defenses such as format-transforming encryption (FTE) [17, 35]. As future work, we plan to investigate using `libFTE` [3] with our design and also empirically evaluate how the use of multiple simultaneous nonces with individual flows may affect the accuracy of traffic analysis tasks.

5.2 Trust in a Centralized Controller

As a consequence of building PHEAR on top of an SDN, we are bound to the requirement for an SDN controller. The controller, which is usually centralized, is a single point of failure and could be a valuable target for attack. Decentralized SDN controllers have been proposed to load balance the control traffic and improve network performance [16]. However, current designs still require that the controller(s) be trusted. Private information retrieval (PIR) may be a promising approach for hiding the SDN queries from the controller(s), thereby reducing the degree of trust in the controller(s), as the performance of information-theoretic PIR schemes is likely sufficient to support online SDN controller queries [13].

5.3 Importance of Network Topology

The security of PHEAR is dependent largely on an adversary's ability to compromise key network switches at the edges of the network. This position enables the attacker to directly observe the end-hosts directly connect to the compromised switch. Consequently, our design would be most promising for networks with a large number of switches connected to clients, and also a large number of "core" switches that connect switches or routers together. Few LAN/WAN-level network topology data sets are publicly available; however, a portion of the Stanford University campus network is described in [31]: this topology consists of ten switches connected to end-hosts and sixteen core routers/switches that connect the edge switches. Such a complex, hierarchical network would be suitable for a PHEAR deployment.

5.4 Road to Deployment

An important practical consideration for any new system is the system's path to real-world deployment. PHEAR leverages the existing network infrastructure (assuming that an SDN infrastructure is already in place) and requires no additional hardware components beyond the standard SDN switches and controller. The end-hosts, however, must run the PHEAR client component in order to use the system.[5] We note that any end-host may opt-in to using PHEAR while those hosts that do not wish to adopt are still able to use the standard MAC and TCP/IP network protocols, as the same network infrastructure can route both PHEAR and non-PHEAR traffic simultaneously. These hosts, however, will not receive the security and privacy benefits offered by our system. As such, our design can be incrementally deployed to any clients who wish to opt-in.[6]

6. RELATED WORK

Tor [15] is the most popular low-latency anonymity network with an estimated two million daily users as of December 2014 [6] and thousands of volunteer-operated Tor routers. Other low-latency anonymity network designs have been proposed [12,20,43,44], but have not been widely adopted. Tor uses a decentralized client/server architecture with layered encryption based on onion routing and is the de facto standard for anonymizing traffic on the Internet. However, Tor is not suitable for protecting traffic that does not leave its origin network and does not transit the public Internet. As such, it is not appropriate for anonymizing traffic within an enterprise or campus network. PHEAR is designed to ensure privacy for precisely this class of traffic.

The IPsec protocol family can ensure the confidentiality, authenticity, and integrity of IP packets [32]. IPsec offers two modes of operation: transport and tunneling mode. In transport mode, the IP payload is encrypted and authenticated point-to-point, but the IP header (which reveals identifying addresses) is still sent in the clear. In tunneling mode, an entire IP packet is encapsulated within an Encapsulating Security Payload. IPsec in tunneling mode is often used to create a virtual private network (VPN), but it requires a dedicated gateway host that may be subject to monitoring by a dedicated adversary. PHEAR leverages IPsec in transport mode to ensure confidentiality of IP payloads (e.g., the transport header and above) while using SDN controller logic to dynamically route IP packets with obfuscated addresses. Incidentally, IPsec has also been proposed as a replacement for TLS in Tor [11].

MACsec provides a connectionless medium access control protocol with confidentiality, integrity, and authentication for Ethernet frames [1]. Standardized as IEEE 802.1AE, MACsec protects the payload of the MAC frame, but it does not hide the MAC addresses that identify the physical network interface. In contrast, PHEAR guarantees that all identifiers present in the MAC header (and higher protocol headers) are removed.

SDN has been proposed as a tool to enhance network security. For example, OpenFlow Random Host Mutation (OF-RHM) has been proposed to protect networks from scanning attacks [27]. OF-RHM works by dynamically translating real IP addresses into short-lived "virtual" IP addresses. Packets with virtual IP addresses are routed dynamically to the correct host using a custom SDN controller. However, OF-RHM distributes virtual IP addresses through standard name resolution services (*e.g.,* DNS), so any host can perform a name lookup to retrieve a target's virtual IP address. Consequently, OF-RHM cannot ensure the anonymity properties that are provided by PHEAR.

7. CONCLUSION

This paper presents the design, implementation, and evaluation of PHEAR, a system built atop existing Software-Defined Networking protocols and standards, that efficiently and transparently removes implicit and explicit identifiers from network traffic. When used in concert with existing IP security protocols (IPsec), PHEAR effectively obfuscates the entire packet from the MAC layer and above. Experiments conducted in a network emulation environment show that our system has sufficiently low latency and high throughput to support common applications including web browsing and file sharing.

8. REFERENCES

[1] 802.1AE-2006 - IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security. `http://standards.ieee.org/findstds/standard/802.1AE-2006.html`.

[2] anonabox. `http://anonabox.com`.

[3] libFTE. `https://libfte.org`.

[5] This is a similar requirement to Tor users running and configuring a local Tor proxy, but the PHEAR client requires super-user privileges to intercept/rewrite packets.

[6] To ease the deployment of the PHEAR clients, we note that the client component could be run at a network gateway or WiFi router, similar to the Torouter [7] or anonabox [2].

[4] Passive OS Fingerprinting. http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting.

[5] TCP/IP Fingerprinting Methods Supported by Nmap. http://nmap.org/book/osdetect-methods.html.

[6] Tor Metrics—Users. https://metrics.torproject.org/users.html.

[7] Torouter. https://trac.torproject.org/projects/tor/wiki/doc/Torouter.

[8] APT1: Exposing One of China's Cyber Espionage Units. http://intelreport.mandiant.com/, 2013.

[9] Operation Cleaver. http://www0.cylance.com/assets/Cleaver/, 2014.

[10] Protecting against misfortune cookie and tr-069 acs vulnerabilities, 2014.

[11] M. AlSabah and I. Goldberg. PCTCP: Per-Circuit TCP-over-IPsec Transport for Anonymous Communication Overlay Networks. In *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)*, November 2013.

[12] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[13] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 13–13, Berkeley, CA, USA, 2012. USENIX Association.

[14] T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[15] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed sdn controller. *SIGCOMM Comput. Commun. Rev.*, 43(4):7–12, Aug. 2013.

[17] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)*, November 2013.

[18] D. Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.

[19] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ACM SIGPLAN Notices*, volume 46, pages 279–291. ACM, 2011.

[20] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, November 2002.

[21] D. Goodin. Bizarre attack infects linksys routers with self-replicating malware, 2014.

[22] N. E. I. S. Group. Network functions virtualisation (nfv) architectural framework. Technical Report ETSI GS NFV 002 V1.2.1, ETSI, December 2014.

[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[24] C. Heffner. Remote attacks against soho routers. In *Blackhat USA*, 2010.

[25] V. D. Izadinia, D. G. Kourie, and J. H. Eloff. Uncovering identities: A study into vpn tunnel fingerprinting. *Computers & Security*, 25(2):97–105, 2006.

[26] D. Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, Apr. 2002.

[27] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[28] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine. Methodically Modeling the Tor Network. In *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2012)*, August 2012.

[29] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 21th ACM conference on Computer and Communications Security (CCS 2014)*, November 2014.

[30] C. Kaufman. Internet key exchange (ikev2) protocol. 2005.

[31] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[32] S. Kent and K. Seo. Security Architecture for the Internet Protocol (RFC 4301). https://tools.ietf.org/html/rfc4301, 2005.

[33] T. Kohno, A. Broido, and k. claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, May 2005.

[34] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In A. Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.

[35] D. Luchaup, K. P. Dyer, S. Jha, T. Ristenpart, and T. Shrimpton. Libfte: A toolkit for constructing practical, format-abiding encryption schemes. In *Proceedings of 23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, August 2014. USENIX Association.

[36] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 90–102, New York, NY, USA, 2009. ACM.

[37] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[38] S. J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of CCS 2006*, November 2006.

[39] T. Nelson, A. D. Ferguson, M. J. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. *NSDI, Apr*, 2014.

[40] Open Networking Foundation. *OpenFlow Switch Specification*, 1.3.4 edition, March 2014.

[41] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 99–110, New York, NY, USA, 2007. ACM.

[42] A. Pfitzmann and M. Hansen. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. *Version v0*, 31:15, 2008.

[43] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.

[44] M. Rennhard and B. Plattner. Practical anonymity for the masses with morphmix. In A. Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 233–250. Springer-Verlag, LNCS 3110, February 2004.

[45] K. Seo and S. Kent. Security architecture for the internet protocol. 2005.

[46] A. Serjantov and P. Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Proceedings of ESORICS 2003*, October 2003.

[47] C. Wacek, H. Tan, K. Bauer, and M. Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Proceedings of the Network and Distributed System Security Symposium - NDSS'13*. Internet Society, February 2013.

[48] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of 23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, August 2014. USENIX Association.

[49] T. Wang and I. Goldberg. Improved website fingerprinting on tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2013)*. ACM, November 2013.

[50] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 3–18, Washington, DC, USA, 2011. IEEE Computer Society.

[51] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 35–49, Washington, DC, USA, 2008. IEEE Computer Society.

[52] C. V. Wright, F. Monrose, and G. M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *Journal of Machine Learning Research*, 7:2745–2769, 2006.

[53] J. Yu and E. Chan-Tin. Identifying webbrowsers in encrypted communications. In *Proceedings of the 12th Workshop on Privacy in the Electronic Society (WPES)*, November 2014.